VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Tomáš Jašica

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Jašica**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: profiq s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek**

Konzultant bakalářské práce: Ing. Rastislav Kanócz

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 11. dubna 2019 .....Jáška.....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 11. dubna 2019

Kováč

profiq s.r.o.
Nad Okrouhlíkem 17/2365
182 00 Praha 8 - Libeň
IČ: 24660761 DIČ: CZ24660761

Chcel by som poďakovať spoločnosti profiq s.r.o za poskytnutie možnosti vykonávať bakalársku prax. Menovite chcem poďakovať konzultantovi zo spoločnosti profiq Ing. Rastislavovi Kanócz a vedúcemu práce Ing. Pavlovi Dohnálkovi za rady a pomoc pri spracovaní tejto práce.

Abstrakt

Táto bakalárska práca popisuje priebeh odbornej praxe v spoločnosti profiq s.r.o. V prvých kapitolách tejto práce je popísaná firma v ktorej študent pracoval, taktiež náplň práce a úlohy, ktoré počas praxe riešil. Ďalej sú popísané technológie, ktoré boli použité pri práci. V ďalšej časti je riešenie daných úloh. Študent mal za úlohu vypracovať dva projekty. Prvým z nich bola implementácia webovej aplikácie pomocou knižnice React v jazyku JavaScript. Následne pracoval na automatizácií virtuálnych strojov pomocou VMware SDK. V práci sú popísané jednotlivé problémy, ktoré vznikly v priebehu vykonávania a spôsoby ich riešenia. Na konci práce sú zhrnuté uplatnené znalosti získané v priebehu štúdia a znalosti, ktoré študentovi chýbali. V závere je popísané celkové hodnotenie odbornej praxe.

Kľúčové slová: odborná prax, JavaScript, Python, React, TeamCity, automatizácia, Cloud Agent, Docker, vSphere, VMware, virtualizácia

Abstract

In this contents, bachelor thesis describes the course of professional practice in profiq s.r.o company. The first part of the thesis describes the company in which the student worked, also the job description and tasks which he dealt during practice. Next is described technologies that were used at work. In the next part is described solution of given tasks. The student had two projects. The first of, was to implement a web application in JavaScript using React library. The second project was to automatization virtual machines with using VMware SDK. Thesis also contains the description of various problems, which risen during the performing of said tasks and their solutions. Towards the end of the thesis is summarized the knowledge gained during the school studies and also the knowledge student was missing. In the end is the overall evaluation of the professional practice.

Key Words: professional practice, JavaScript, Python, React, TeamCity, automatization, Cloud Agent, Docker, vSphere, VMware, virtualization

Obsah

Zoznam použitých skratiek a symbolov

MVC	–	Model View Controller
VMFS	–	Virtual Machine File System
API	–	Application Programming Interface
GUI	–	Graphical User Interface
TC	–	TeamCity
CSS	–	Cascade Styling Sheets
HTML	–	Hypertext Markup Language
AQL	–	Artifactory Query Language
JSON	–	JavaScript Object Notation
SVN	–	Subversion
CI	–	Continuous Integration
HTTP	–	Hypertext Transfer Protocol

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Obsah tejto bakalárskej práce popisuje moju odbornú prax v spoločnosti profiq s.r.o. "Táto firma sa v dobe môjho nástupu venovala predovšetkým poskytovaniu služieb, v oblasti vývoja a testovania softwaru, agilným softwarovým spoločnostiam zo Silicon Valley ale aj inych častí US. Poskytuje zaujímavé projekty v oblastiach tzv. full-stack vývoja alebo automatizácie testovania a integrácie takej automatizácie do priebežného build procesu (tzv. continues integration).

V tomto dokumente popisujem prácu, ktorú som vykonával v rámci mojej bakalárskej praxe. Po nástupe do firmy bolo mojou úlohou zoznámiť sa s jazykom JavaScript a knižnicou React. Pomocou tejto knižnice som mal vytvoriť frontend pre webovú aplikáciu a následne za pomoci GitLab CI zautamatizovať build proces tak aby nasadil túto aplikáciu na server s využitím technológie Docker. Po dokončení prvej úlohy som sa presunul k druhému projektu, automatizácií správy virtuálných strojov pomocou WMWare SDK pre jazyk Python. V tejto úlohe som sa najskôr musel naučiť pracovať s virtuálnymi strojmi, správou TeamCity serveru a rôzných technológií, popísaných v kapitole 3 na strane 13. Hlavnou úlohou bolo vyriešiť automatické ukladanie balíčkových systémov na virtuálnych strojoch (tzv. cloud agentoch), ktoré sa automaticky alokovali pomocou TeamCity serveru. Táto úloha bola trošku obecnejšia a obsahovala rôzne úskalia, kde sme použili množstvo technológií a postupným riešením práce sme prišli na lepšie riešenie.

Táto práca ma obmedzený rozsah dokumentu, nebolo preto možné popísať všetky znalosti, ktoré som sa naučil a preto sa predovšetkým zameriavam na technológie, ktoré som sa v priebehu odbornej praxe musel naučiť a na popis, ako som tieto technológie využil k dosiahnutiu požadovaného výsledku.

Na tejto práci a firmou zadaných projektoch som pracoval sám za pomoci môjho vedúceho. Vo firme som následne pokračoval na ďalších, rôznych projektoch, avšak ako bežný zamestnanec.

2 Popis firmy a pracovná náplň študenta

V tejto kapitole sa venujem popisu firmy v ktorej som pracoval a náplní práce, ktorú som počas praxe vykonával.

2.1 profiq s.r.o

Firma profiq, sídliaca v Českej republike, poskytuje služby pre softwarové spoločnosti v Silicon Valley, ktoré vyvíjajú software agilným spôsobom. Služby spoločnosti profiq zahŕňajú testovanie, vývoj, a v neposlednom rade údržbu softwaru.

Spomínaná spoločnosť profiq s.r.o v dobe písania zamestnáva vyše 50 zamestnancov, čo z nej vytvára stredný podnik. Firma ponúka nielen odborníkov na jednotlivé úlohy ale aj celé tímy pre komplexné riešenie daných projektov. Firma je najrozšírenejšia v oblasti webových a mobilných služieb, zabezpečenia a cloudu.

Spoločnosť si prešla radom zákazníkov, ktoré sú svetovo známe a je ich už viac než 10. V čase mojej praxe vo firme sme spolupracovali s firmami ako napr. Liferay, Avast, ForgeRock, Slamdata, DivvyPay alebo Globiq.

2.2 Pracovná náplň študenta

Firma ma prijala na pozíciu softwarového inžiniera. V spolupráci so školou a odbornou praxou vo firme mi bol priradený projekt na dokončenie webovej aplikácie Statuspage, kde som v React knižnici vytváral frontend a pomocou GitLabu CI a dockeru nasadil aplikáciu na server. Význam tejto webovej aplikácie a implementáciu pomocou spomínaných technológií popisujem v mojej práci v kapitole 3.

Hneď po dokončení som sa teda presunul k druhej, oveľa komplexnejšej úlohe kde som pomocou vmware SDK mal zautomatizovať správu virtuálných strojov a pravidelnú aktualizáciu. Projekt bol určený pre všetky druhy virtuálnych strojov a preto sme sa stretli s rôznymi operačnými systémami, od Windows cez rôzne verzie Linuxu až po OS X. V tomto projekte som sa musel naučiť pracovať nielen s doposiaľ nadobudnutými skúsenosťami ale naučiť sa dobre ovládať TeamCity server a Artifactory. Pri postupe sme taktiež používali jazyky ako AQL (Artifactory query language) alebo Gradle.

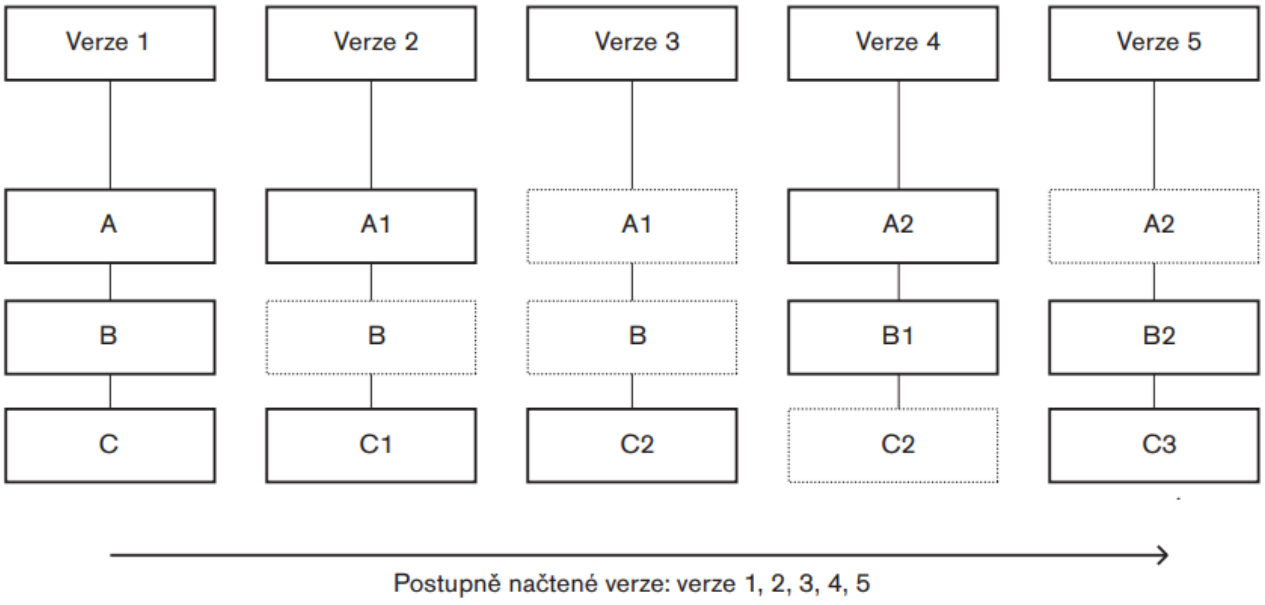
3 Použité technológie

Táto kapitola sa venuje stručnému popisu najčastejšie používaných aplikácií. Ďalšie kaptioly v tejto práci sa na tieto technológie odkazujú.

3.1 GIT

Git je systém pre správu zdrojových kódov. Správa verzií je systém, ktorý zaznamenáva zmeny alebo sady súborov v priebehu času a užívateľ tak môže kedykoľvek obnoviť jeho konkrétnu verziu.

Hlavným rozdielom medzi systémom Git a ostatnými verzovacími systémami je že data spracuváva inak. A to tak, že ich ukladá ako sadu snímkov vlastného, malého systému súborov. Vždy, keď v systéme uložíme stav projektu, Git v podstate vytvorí snímok, ako vyzerajú ostatné súbory v danom okamžiku a uloží referencie na tento snímok. Ak v súboroch neboli prevedené žiadne zmeny, Git v záujme zefektívnenia práce neukladá celý súbor, ale len odkaz na predchádzajúci identický súbor, ktorý už bol uložený.[1] Pomocou tohto systému, narozdiel od ostatných verzovacích systémov, šetríme veľké množstvo úložiska, ktoré je v tejto dobe veľkým problémom. Spracovanie dát ilustruje obrázok 1.



Obr. 1: Ukážka ukladania verzií pomocou Git

3.2 React

React je populárna knižnica používana pre vytvorenie užívateľského rozhrania. Vytvorila ju spoločnosť Facebook aby vyriešila niektoré výzvy spojené s rozsiahlými webovými stránkami s veľkým množstvom dát. [3]

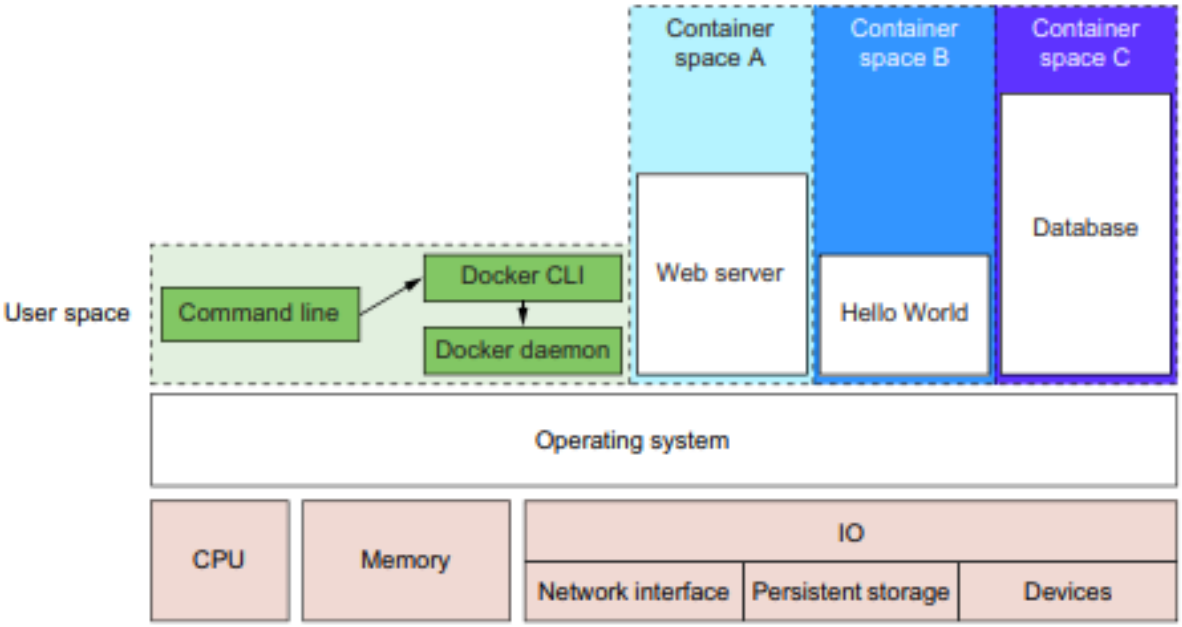
React knižnica je založená na komponentách, ktoré sú nezávislé časti aplikácie a môžme ich opakovane použiť.

Môžme ich zlučovať a skladať do iných komponent čím vytvoríme výslednú aplikáciu. V Reacte teda iba popíšeme ako má vyzerať výsledná stránka na základe predchádzajúcich dát a našou úlohou je potom mu dodávať nové data do jednotlivých komponent.

3.3 Docker

Docker je program príkazového riadku, daemon na pozadí a sada vzdialených služieb. Využíva logistický prístup k riešeniu bežných problémov so softvérom a zjednodušuje inštaláciu, spustenie, publikovanie a odstránenie. Všetko pomocou technológie UNIX nazývanou kontajnery. [4]

Docker môžeme využiť ako náhradu virtuálnych strojov, ktoré nie sú tak efektívne keďže sa dlho štartujú. Docker však nepoužíva hardwarovú virtualizáciu a programy v ňom spustené bežia v kontajneroch. Medzi programom bežiacim v kontajneri a operačným systémom není žiadna vrstva, neplýtvame žiadným softvérom alebo simuláciou virtuálneho hardvéru. Docker teda neni virtualizačná technológia, ale pomáha nám používaním kontajnerov, ktoré sú zabudované v našom operačnom systéme.



Obr. 2: Ukážka troch kontajnerov bežiacich na operačnom systéme Linux

3.4 VMware vSphere

VMware vSphere spája niekoľko odlišných produktov a technológií, ktoré spolu poskytujú kompletnú infraštruktúru pre virtualizáciu. vSphere podporuje prechod existujúcich data centier do verejných cloudových centier. Jeho nevýhodou však je, že táto technológia je pomerne drahá a potrebuje skúsenú administráciu.
Jeho technolģie sú rozdelené do niekoľkých produktov:

- **ESXi** - je jádrom a hypervisorom vSphere, ktorý beži na hosťovských počitačoch, aby riadil správu virtuálnych počítačov a presúval výkon do virtualných strojov podľa potreby.[5]
- **vCenter Server** - je serverová aplikácia, bežiaca na Windows Servery vo virtuálnom stroji. Je centrálnym bodom pre vytvorenie, štartovanie a ostatnú správu virtuálných strojov.[5]
- **vCenter Client** - je aplikácia pre vzdialený prístup k funkciam serveru vCenter. Je to najviac používaný nástroj pri správe celého vSphere prostredia.[5]
- **VMFS** - je to súborový systém pre správu diskov, ktoré sú k dispozícií virtuálným strojom[5]

3.5 JFrog Artifactory

Artifactory je manažér repozitárov, ktorý podporuje všetky dostupné typy softwarevých balíčkov. Je navrhnutý tak aby sa integroval s väčšinou nástrojov pre nepretržitú integráciu a dodávku a aby bol zaistený automatický dohľad od vývoja až po produkciu.[6]

Tento manažér nám prináša veľkú škálu funkcií, ktoré sú pre rozsiahlejšie projekty nevyhnutné. Stará sa nie len o automatizovaný prístup k balíčkom zo vzdialených strojov ale aj o bezpečnosť, ktorú môžme spravovať použitím užívateľov a skupín, ktorým môžme priradiť určité práva. Artifactory sa za nás pravidelne postará o zálohovanie knižníc a všetky možnosti môžme administrovať pomocou webového rozhrania, ktoré nám Artifactory ponúka.

3.6 TeamCity

TeamCity je výkonný, kontinuálne integrovaný a užívateľsky priateľsky server, ktorý pracuje mimo nás.[7] Tento server umožňuje vývojárským týmom monitorovať proces počas zostavenia a vylepšiť týmovú prácu. V TeamCity projektoch môžme nielen vytvárať kód ale taktiež ho testovať, ladiť alebo nasadzovať. Pomocou neho máme možnosť sledovať výstupy alebo odchytávať chyby spoločných projektov. Server podporuje ľubovoľné programovacie jazyky, takže ho môžme použiť od Javy až po mobilné platformy.

TeamCity drží pod správou virtuálne alebo fyzické stroje, na ktoré sa pripája pomocou TeamCity agenta a zostavenie projektu prebieha na tomto, hosťujúcom zariadení.

3.7 Gradle

Gradle je nástroj pre automatizáciu. Najčastejšie sa používa na zostavenie nejakého projektu. Je kontinuálne integrovaný a dokáže taktiež sprevádzkovať nasadenie alebo generovať dokumentáciu.

Gradle umožnuje zostavu akéhokoľvek softwaru, pretože robí len málo predpokladov o tom, čo chcete zostaviť alebo ako by to malo byť. Obmedzením v tomto prípade je, že v súčastnosti podporuje len repozitáre Maven a systémové súbory Ivy. [8]

Pre zostavenie využíva balíčkový systém, ktoré sú ako závislosti pre zostavenie daného projektu. Tieto závislosti sú uložené v maven repozitári, ktorý nám umožňuje pristupovať k ním veľmi jednoducho.

4 Zoznam úloh zadaných v priebehu odbornej praxe

Táto časť bakalárskej práce popisuje úlohy zadané v priebehu odbornej praxe a ich následné riešenie, postupne ako boli vykonané.

4.1 Webová aplikácia Status Page

4.1.1 Popis produktu

Webová aplikácia Status Page je komunikačný nástroj na monitorovanie a upozorňovanie tímov, zákazníkov alebo používateľov. Táto webová aplikácia monitoruje váš, prípadne zákazníkov produkt v reálnom čase. Produktom myslíme webovú stránku alebo server. Ak je produkt nedostupný alebo má problémy so spojením, aplikácia upozorní svojich užívateľov pomocou e-mailu. V databáze si uchováva data, kedy a či bol produkt dostupný. Z týchto dát môžme následne vytvoriť analýzu alebo napríklad grafy o dostupnosti daného produktu za určitý čas.

Jej funkčnosť spočíva v tom, že zákazník alebo užívateľ si môže pozrieť či je problém so spojením na jeho strane alebo je za tým nejaká chyba. Správca produktu môže po upozornení výpadku pomocou tejto stránky oznámiť svojím uživateľom, či sa výpadok vyšetruje, rieši alebo je už vyriešený.

4.1.2 Návrh a implementácia GUI

Keďže našou prioritou bolo aby sa stránka podobala webovej stránke spoločnosti profiq s.r.o. a spĺňala všetko potrebné, teda aby bola pekná a mala užívateľsky priateľský vzhľad, inšpiroval som sa vzormi a návrhmi nájdenými na webe. Po spoločnej diskusii sme vybrali jeden návrh a upravili podľa našej potreby. Keďže to nebola zložitá úprava, postačilo vytlačiť daný návrh a zakresliť zmeny.Pre implementáciu sme zvolili programovací jazyk JavaScript a knižnicu React.

Programovací jazyk JavaScript je multiplatformný, objektovo orientovaný, skriptvací jazyk. Najčastejšie sa používa pri implementácií webových stránok, kde je vkladaný ako súčasť HTML kódu stránky. Pomocou neho obvykle ovládame rôzne interaktívne prvky ako tlačidla, textové polia. Taktiež môžme v ňom vytvárať animácie alebo efekty obrázkov. Program v JavaScripte sa zvyčajne púšťa až po stiahnutí stránky z internet a beží na strane klienta a nezaťažuje tým server.

Zmeny v implementácií som mal posielať na GitLab, kde mal môj tím uložené zdrojové kódy. Keďže môj kolega pre mňa vytvoril repozitár v rámci serveru Git, stiahol som si tento projekt pomocou Git príkazu

```
clone https://gitlab.profiq.com/profiq-infra/ldap/frontend
```

Ak som chcel zistiť stav jednotlivých súborov, použil som príkaz *git status*. Najčastejšie som využíval príkaz *git commit*[1], ku ktorému som pomocou parametru *-m* pridal komentár, aké zmeny

[1]Príkaz, pomocou ktorého zapíšeme zmenené súbory

som od poslednej verzie vykonal. Ak som vytvoril nejakú prácu, ktorú som chcel poslať vedúcemu, použil som príkaz *git push*, ktorý automaticky pošle moje verzie (commity) do vzdialeného repozitára. Ďalej som mohol vytvoriť žiadosť o zaradenie kódu do hlavnej vetvy *pull request*[2]. Túto žiadosť môj vedúci schválil alebo mi vrátil daný kód vrátane poznámok, čo je potreba zmeniť. Ak bolo všetko v poriadku, Git zlúčil tento kód do hlavnej vetvy.

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/master'.
#
# Changes to be committed:
#       modified:  src/App.js
```

Výpis 1: Ukážka editoru po spustení príkazu commit.

Ako bolo spomínané v kapitole 3.2, React je postavený na myšlienke že všetky časti sú komponenty, ktoré môžme do seba nezávislé nasádzať. Aby som mohol v našej aplikácií použiť balíčky dostupné na internete, vytvoril som náš projekt pomocou správcu javascriptových balíčkov *npm*. Pomocou tohto správcu som pridával balíčky voľne dostupné na internete, ktoré obsahujú už vytvorené komponenty. Vybral som si balíčkek Ant Design a nasadil do našej aplikácie pomocou príkazu

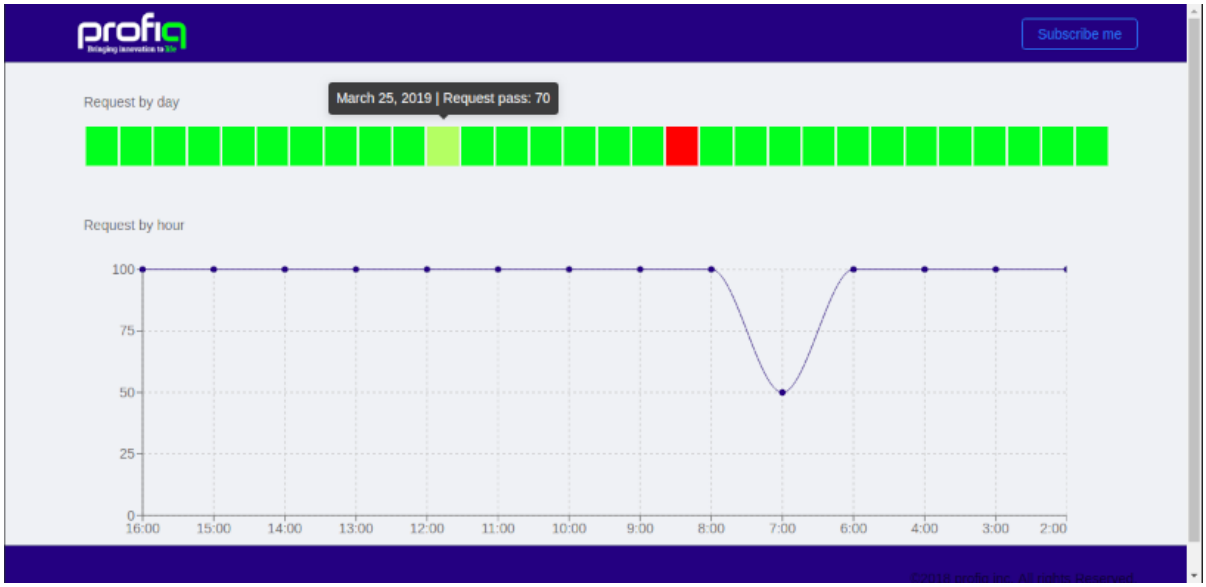
```
npm install antd
```

Vytvoril som si rozloženie stránky a do neho nasadil ostatné komponenty.

Pre vizualizáciu histórie za posledných 30 dní som použil základné html príkazy s úpravou v CSS súbore, v ktorom som si definoval element *.rect*. Nastavil som mu výšku, šírku a medzeru medzi každým z nich aby boli rozlíšiteľné. Vo finále to bol malý obdĺžnik predstavujúci jeden deň. Tento element som volal v komponente Reactu pomocou cyklu tak, aby nám ho zobrazilo 30 krát a každému z nich som priradil špecifický kľúč pomocou ktorého som mu mohol pridávať hodnoty a podľa toho vyfarbiť. Kolega vytvoril aplikáciu, ktorá ukladala aktuálne data do databázy a ja som ich odtiaľto pomocou HTTP metódy GET mohol získať. Vytvoril som škálu farieb a priraďoval ich špecifickému elementu na základe konkrétnych dát. Na rozlíšenie farieb podľa určitých dát som použil základné podmienky jazyku JavaScript.

Taktiež sme chceli do stránky zakomponovať graf, ktorý ale knižnica Ant Design neponúka. Nasadil som teda druhý balíček Recharts a vybral jednoduchý čiarový graf, ktorý som upravil tak aby získaval taktiež aktuálne hodnoty z databáze. Hodnoty pre vizualizáciu histórie som získaval z databáze, vytvorenou kolegom za pomoci Django. Tieto dáta som získaval pomocou API žiadostí.

[2]Žiadosť pre správcu projektu o zaradenie kódu



Obr. 3: Ukážka webovej stránky

4.1.3 Docker

Po dokončení implementácie stránky v Reacte som našu aplikáciu mal spúšťať pomocou Docker kontajneru.

Aby nám Docker takýto kontajner zostavil, vytvoril som v destinácií nášho projektu konfiguračný súbor, v ktorom som zadal potrebné kroky k tomu aby nám Docker mohol vytvoriť obraz a zanechal v ňom potrebné súbory a spúšťal príkazy pre spustenie nášho React projektu pomocou balíčkového systému npm [3].

Keď všetko prebehlo úspešne, spustil som tento obraz pomocou príkazu

```
docker build
```

a overil či stránka funguje lokálne v prehliadači. Po dokončení týchto úloh som všetky súbory nahral pomocou git pull[4] do nášho repozitára na GitLab.

4.1.4 GitLab CI

Pre automatizáciu zostavenia, má GitLab svoj nástroj *GitLab Continuous Integration* (CI), ktorý funguje na podobnom princípe ako TeamCity. Aby nám nástroj CI fungoval podľa naších požiadaviek, vytvoril som v našom repozitári konfiguračný súbor *.gitlab-ci.yml*. Tento súbor konfiguruje čo sa má stať s naším projektom a akceptuje rôzne parametre. Pre mňa bolo dôležité zostaviť našu aplikáciu pomocou Docker obrazu, ktorý nám tento nástroj umožňuje. Ďalej sme nastavili kroky, ktoré sa vykonajú pred samotným zostavením a daný projekt spustíme. Takýchto

[3] Správca JavaScriptových balíčkov
[4] Stiahne commity a prevedie zlúčenie verzií

krokov môžme použiť viac a taktiež môžme v konfigurácií nastaviť automatické testovanie pred spustením.

4.2 Automatizácia správy virtuálných strojov

V tejto kapitole sa budem zaoberať celým procesom, od popisu výsledného produktu až po jeho reálne použitie.

4.2.1 Popis produktu

Druhou mojou úlohou odbornej praxe bolo automatizovať správu virtuálnych strojov pomocou vmWare SDK. V mojom prípade sa jednalo o správu strojov ktoré používal server TeamCity. Tieto stroje pomerne dosť času strávia odstavené a nemajú prácu, pričom aj tak zaberajú veľkú časť výkonu v data centrách. Pomocou kompatibility s TeamCity sme preto automatizovali životnosť týchto strojov a pomocou vmWareSDK zautomatizovali aktualizáciu systému, aktualizáciu agenta a pravidelne nahrávali Gradle cache potrebnú pre Gradle zostavenia.

TeamCity okrem klasických agentov, vďaka kompatibilite s vmWare umožňuje použiť tzv. cloud agentov, ktorí sa starajú aby stroje vo vSphere nevyužívali nevyužitý výkon a po dokončení projektu sú vypnuté a vymazané. Ak je spustený nový projekt a agent je potrebný, TeamCity dá pokyn do vmWare aby alokoval novú mašinu, kde sa daný projekt zostaví.
Postup práce som si teda zhrnul takto:

1. Vytvoriť virtuálny stroj a nainštalovať potrebné veci pre správny chod.
2. Nastaviť konfiguráciu TeamCity serveru pre cloud agenta.
3. Vytvoriť skript a pomocou vmWare SDK aktualizovať a spravovať tento stroj.
4. Pomocou AQL získať z Artifactory štatistiky o balíčkoch pre Gradle.
5. Vytvoriť konfiguračný súbor pre Gradle aby najsťahovanejšie balíčky skompiloval do vyrovnavacej pamäti na stroji.
6. V TeamCity vytvoriť projekt, ktorý bude všetko automaticky spúšťať bez nášho zásahu

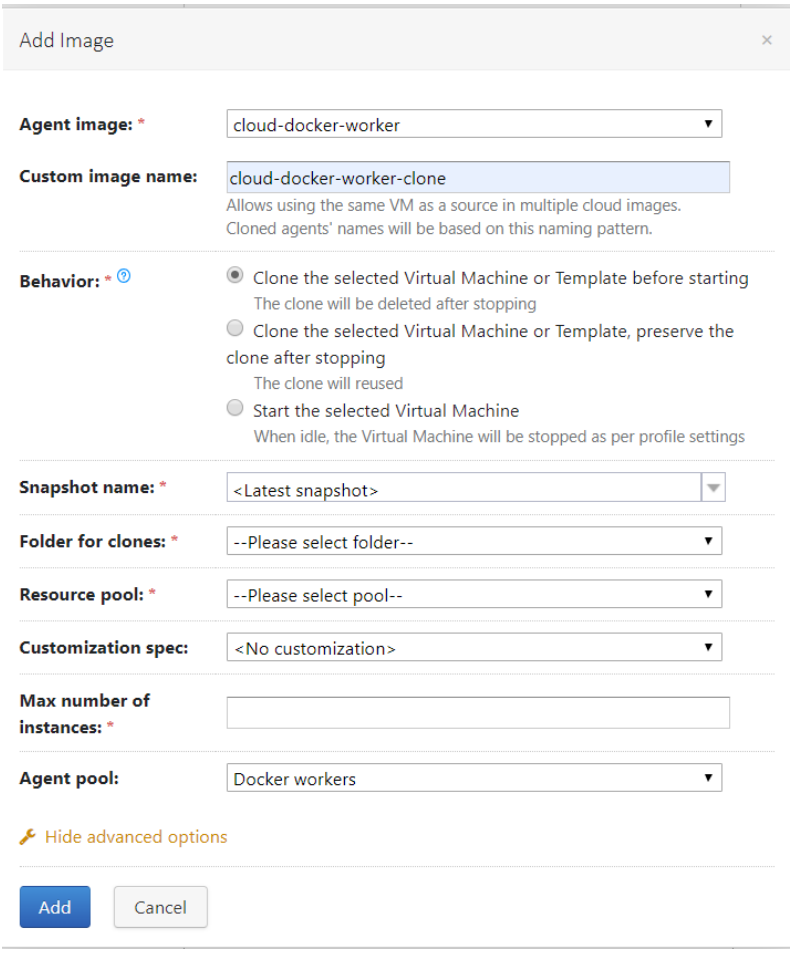
4.2.2 Vytvorenie TeamCity Cloud Agentov

Ako prvé som teda potreboval vytvoriť cloud agentov. Aby som takýchto cloud agentov vytvoril, potreboval som vzorový stroj uložený vo vmWare, z ktorého bude TeamCity vytvárať nové, rovnaké stroje ako je vzor. Vztvoril som si teda nový stroj, s takým výkonom aby spĺňal hardwerovú záťaž všetkých predpokladných zostavení. Na stroj som nainštaloval všetok potrebný software aby mohli na ňom prebehnúť všetky zostavenia bez akýchkoľvek problémov a pre spojenie s TeamCity som na stroj nainštaloval TeamCity agenta, ktorý je rovnaký ako pri bežne používaných agentoch a nastavil automatické spustenie. Na virtuálnych strojoch môžme taktiež ukladať

aktuálny stav ako verzie stroja. Keď bol stroj pripravený, mohol som ho vypnúť a uložiť novú verziu. V prípade potreby sa môžme vrátiť a spustiť stroj v danej verzii.

V konfigurácií TeamCity som vytvoril profil cloud agenta, ktorý som zaradil k projektom môjho tímu. V nastaveniach profilu som musel okrem názvu a popisu vybrať kompatibilitu s vSphere. Po zadaní prihlasovacích údajov sa mi vytvoril profil, pripojený na vmWare pomocou môjho účtu.

V profile som vytvoril novú šablónu, kde zadáme aký stroj chceme vytvárať. Podstatnou vecou v tomto kroku je zvoliť, ktorú verziu nášho stroja chceme. Ja som zvolil možnosť poslednej verzie čo nám zaručí že ak niečo zmeníme na vzorovom stroji a uložíme novú verziu, tak nemusíme v TeamCity už nič prenastavovať a stroje sa budú automaticky vytvárať z najnovšej verzie. Profil a šablónu uložíme a ak je všetko pripravené, cloud agent sa nám zobrazí ako kompatibilný s naším projektom a môžme ho používať.



Obr. 4: Ukážka pridania vzorového obrazu v TeamCity

4.2.3 vmWare SDK

Stroje vytvorené ako cloud agenti sú však duplikátmi vzorového stroja. Ak teda chceme niečo zmeniť, musíme upraviť našu šablónu a uložiť novú verziu. Avšak, každou aktualizáciou TeamCity serveru sa taktiež aktualizuje TeamCity agent uložený na stroji s ktorým pracuje. Za normálnych okolností sa táto operácia vykoná automaticky po aktualizácií. V našom prípade však máme vzorovú mašinu vypnutú a na novo vytvorených agentoch je stále stará verzia a pri každom alokovaní nového stroja, v našom prípade pri každom zostavení, sa musí najskôr vykonať aktualizácia agenta, ktorá zaberá pomerne veľa času.

Ďalšou mojou úlohou teda bolo zautamtizovať tieto problémy. Ako najlepšia voľba pre túto úlohu bolo zvoliť ovládanie vClienta pomocou API. V tomto prípade sme teda použili vmWare SDK a API knižnicu *pyvmomi* pre programovací jazyk Python, v ktorej môžme ovládať vClienta pomocou príkazov. Python je moderný programovací jazyk, ktorý sa dá jednoducho naučiť no napriek tomu ponúka mnoho typov a štruktúr. Dajú sa v ňom písať veľké a rozsiahle projekty. Tento jazyk je veľmi prehľadný a môžme ho použiť takmer na všetko. V mojom prípade som ho použil na spustiteľné skripty.

Založili sme teda nový projekt a podobným spôsobom ako opisujem v kapitole 4.1.2 som ho spojil so serverom GIT. Prvou dôležitou vecou bolo pripojenie sa do vmWare vCenter Clienta aby tam mohli prebiehať dalšie úkony. Vytvorili sme preto funkciu, ktorá brala potrebné parametre ako webovú adresu vClienta a prihlasovacie údaje.

```
def connect_to_vc(args):
   si = None
   try:
       context = None
       if hasattr(ssl, '_create_unverified_context'):
           context = ssl._create_unverified_context()
       si = SmartConnect(host=args.host, user=args.user,pwd=args.password,
                   port=int(args.port),sslContext=context)
       if not si:
           sys.exit()
       atexit.register(Disconnect, si)
       return si
   except vmodl.MethodFault as e:
       print("Caught vmodl fault : " + e.msg)
   except Exception as e:
       print("Caught Exception : " + str(e))
   return si
```

Výpis 2: Funkcia pre pripojenie do vCenter Client

Po pripojení potrebujeme aby sa náš vzorový stroj zapol. K tomu sme použili funkciu *PowerOn()*[5], ktorá je súčasťou knižnice *pyVmomi*[6]. Po zapnutí si overíme, aký má daný stroj operačný systém. Ak je linuxový, spustíme na ňom aktualizácie systému z príkazového riadku. V systéme Ubuntu pomocou príkazu *sudo apt-get update -y* a v CentOS pomocou *sudo yum update -y*

Ďalšou, pre nás dôležitou úlohou bolo stiahnúť, prípadne aktualizovať Gradle Cache, ktorú opisujem v kapitole 4.2.5 aby sa nemuseli závislosti sťahovať pri každom zostavení projektu a ušetrili sme tým čas. Ak všetky aktualizácie prebehnú v poriadku, stroj sa vypne a vytvorí nová verzia, ktorá sa prejaví pri ďalších zostaveniach.

4.2.4 Artifactory

Aby som urýchlil proces zostavenia pre Gradle projekty, potreboval som stiahnúť, už spomínané závislosti pre tento projekt. Keďže náš stroj je kompatibilný s rôznymi projektmi a každý projekt používa špecifické závislosti, potrebovali sme tých závislosti čo najviac. Problém je, že nemáme neobmedzené množstvo pamäti a preto potrebujeme stiahnúť len tie najdôležitejšie. Náš server beží už niekoľko rokov a preto sú pre nás tieto štatistiky použiteľné. V našom prípade Gradle sťahuje tieto závislosti z maven repozitára, ktorý spravuje server JFrog Artifactory. Ten nielen spravuje repozitáre ale taktiež si vytvára špecifickú štatistiku a históriu všetkých súborov. K tomu aby som k takejto štatistike mohol pristúpiť a filtrovať konkrétne súbory nám služba Artifactory ponúka API a vlastný dotazovací jazyk Artifactory Query Language (AQL).

V našom dotaze posielame prihlasovacie údaje, ktoré nechceme aby boli viditeľné. Na získanie dát používame HTTP metódy *POST a GET*. Metóda GET posiela nami zadané údaje priamo v URL adrese, čo by z hľadiska bezpečnosti nebolo prijateľné, preto som sa rozhodol pre metódu POST, ktorá tento problém rieši a zadané data niesú viditeľné.

Pomocou HTTP metódy POST (API) som sa teda pripojil na Artifactory server a poslaním AQL dotazu som dostal všetky sťahované súbory, ktoré boli stiahnuté v posledných 3 mesiacoch z repozitára *maven* a taktiež počet celkových stiahnutí daného súboru. K spojeniu funkcie *API POST* a *AQL* som použil jazyk Python a pre ľahšiu manipuláciu s *POST* dotazom som použil knižnicu *requests*. Výsledne data mi API odovzdá vo formáte JavaScript Object Notation (JSON).

```
def stat_download(args):
    username = args.user
    passw = args.passw
    url = "https://artfct.profiq.com/artifactory/api/aql"
    aql = 'items.find({"$and": [{"repo" : "maven",' \
          '"stat.downloaded" : {"$last" : "3mo"}' \
          '}]}).include ( "name","repo", "path", "stat.downloads", "size")'.
    response = requests.post(url, data=aql, auth=(username, passw))
```

[5]Funkcia knižnice pyVmomi
[6]Knižnica v jazyku Python pre správu vmWare pomocou API

```
    data = json.loads(response.text)
    return data
```

Výpis 3: POST dotaz pre stiahnutie štatistík

Repozitár maven je ale veľký a obsahuje niekoľko tisíc súborov. Na to by nám nepostačilo miesto na disku, ktoré máme k dispozícií. Ďalším problémom je, že nie všetky súbory sú balíčky, ktoré potrebujeme pre Gradle projekt ako závislosti. Preto som zo zoznamu filtroval len názvy, ktoré sa skladajú z 5 častí, oddelenými bodkou a dvojbodkou. Všetky takto získané balíčky som zoradil od najsťahovanejších a počet týchto balíčkov zredukoval na prvých 500.

4.2.5 Gradle cache

Gradle cache nám umožňuje zostavenie Gradle projektu zrýchliť a tým získať ďalší čas, keďže není potrebné sťahovať artefakty pri každom zostavení. Táto cache sa kompiluje čo znamená že zdrojový kód sa preloží do vhodnej podoby pre Gradle strojový kód. Takto skompilovaný zdrojový kód sa ukladá lokálne na stroj. Každý projekt teda k nej má prístup a môže ju použiť.

Aby som zabezpečil správne uloženie a kompiláciu týchto závislosti, vytvoril som jednoduchý Gradle projekt, ktorý nerobí nič iné, len stiahne závislosti a uloží ich na svoje miesto, tam kde potrebuje. K tomu som potreboval upraviť konfiguračný súbor *build.gradle* a vložiť do neho všetky názvy balíčkov, ktoré som dostal a filtroval v kapitole 4.2.4. Spojil som teda python funkcie, ktoré mi nielen dostali ale aj zoradili balíčky a pridal som upravenie konfiguračného súboru. Konfiguračný súbor je tak jednoduchý, že som ho generoval priamo v skripte. Príklad výsledného súboru s jedným názvom balíčku je zobrazený vo výpise 4. V priebehu práce sme však prišli na problém, že ak budú balíčky uložené na stroji a nebudú sa sťahovať, časom nebudú najsťahovanejšie a nám sa na stroj dostanú balíčky s menšou prioritou. Preto sme pred spustením Gradle najskôr vymazali všetky staré balíčky aby sme ich mohli nahradiť novými.

```
def print_gradle_build(result):
    gradleFile = open("build.gradle", "w")

    gradleFile.write("apply plugin: 'java'" + '\n')
    gradleFile.write('\n' + "repositories {" + '\n' +
      "maven { url 'https://artfct.profiq.com/maven' }" +
      '\n' + "}" + '\n')
    gradleFile.write("dependencies {" + '\n' )
    for i in range(0, len(result)):
      gradleFile.write(result[i] + '\n')
    gradleFile.write("}")
```

Výpis 4: Funkcia na úpravu konfiguračného súboru.

```
apply plugin: 'java'
repositories {
    mavenCentral()
}
dependencies {
    compile 'org.apache.httpcomponents:httpclient:4.4'
}
```

Výpis 5: Ukážka konfiguračného súboru *build.gradle*

4.3 Nasadenie projektu do TeamCity

Doteraz vytvorené skripty som otestoval na testovacom stroji a keďže všetko fungovalo, odoslal som náš projekt vo finálnej verzií na Git repozitár. V TeamCity som si vytvoril projekt, ktorý pomocou mnou zadaných krokov sťahoval na svoj hosťujúci stroj projekt zo vzdialeného repozitára Git a spustil postupne naše skripty. Pred tým ako sme zadali určité kroky sme si nastavili parametre, ktoré budeme posielať ako argumenty pre naše skripty. Okrem autorizačných argumentov bol jeden hlavný a to stroje, ktoré chceme aktualizovať.

S použitím parametrov sme vytvorili prvý krok aby sa nám spustili vzorové stroje, na ktorých budú prebiehať naše skripty. Keď sa stroje spustili, nahral sa naň náš projekt zo vzdialeného repozitára, spustila sa aktualizácia systému a vymazali staré balíčky uložené na stroji. Následne sa spustil Gradle projekt, ktorý nám na stroj stiahol najsťahovanejšie balíčky, stiahnuté za posledné 3 mesiace. Ak sa ukončili všetky tieto úlohy, stroje sme vypli a vytvorili novú verziu obrazu, ktorá sa prejaví pri ďalších zostaveniach.

5 Znalosti získané v priebehu štúdia a uplatnenie v praxi

V priebehu absolvovania mojej odbornej praxe som uplatnil široké spektrum nadobudnutých informácií počas štúdia na škole a to hlavne v rámci konceptov a štýle programovania, s pomocou ktorých som mohol udržiavať svoj kód prehľadný a rozšíriteľný. V značnej miere mi pomohli taktiež projekty a prednášky z predmetu Tvorba aplikácií pre mobilné zariadenia, kde sme využívali JavaScript a predmet užívateľské rozhranie kde sme pracovali s programovacím jazykom Python, ktorý som v rámci praxe používal najčastejšie.

Ďalšími znalosťami som disponoval z predmetu Počítačové siete, kde som sa naučil konfigurovať nastavenie siete pomocou vzdialeného prístupu v príkazovom riadku. Využil som taktiež informácie z predmetu softwarové inžinierstvo, podľa ktorého som postupoval pri plánovaní a rozvrhnutí projektov.

6 Chýbajúce vedomosti a znalosti v priebehu praxe

Vedomosti, ktoré mi chýbali na praxi je naozaj veľa keďže projekt s ktorým som sa stretol, obsahoval veľké množstvo technológií.

Počas štúdia som sa nestretol so správou niektorého systému správy verzií ako je Git alebo SVN. Vo svojích projektoch som taktiež musel pozdvihnúť svoje informácie v oblasti jazyka JavaScript a Python, ktorý sa na škole preberá skôr na základnej úrovni. Na škole by som ocenil nejaké priblíženie technológie Docker a virtualizácie, ktorá sa používa čoraz častejšie. V druhej fáze bakalárskej praxe som sa musel naučiť pracovať s rôznymi technológiami ako Artifactory, TeamCity, Gradle a Maven.

7 Dosiahnuté výsledky a celkové zhodnotenie odbornej praxe

7.1 Výsledky práce

Výsledkom mojej práce v rámci rozsahu bakalárskej práce boli dva projekty. V prvom som sa zaoberal implementáciou webovej aplikácie. Webovú aplikáciu, ktorá mala za úlohu získavať informácie a overovať dostupnosť stránky našej spoločnosti sme implementovali pomocou jazyka JavaScript a knižnicou React. Implementácia taktiež obsahovala prepojenie užívateľského rozhrania s databázou, ktorá získavala aktuálne informácie potrebné pre vizualizáciu stavu našej stránky. Aby sme takúto stránku mohli jednoducho použiť, využil som technológiu Docker a webovú aplikáciu som nasadil na server pomocou nástroja GitLab CI.

Moju prácu v druhom projekte sme využili na automatickú správu virtuálnych strojov bežiacich vo vSphere. Naše data centrá sme týmto spôsobom odľahčili na minimálnu úroveň a taktiež sme sa postarali o to aby sme nemuseli manuálne zasahovať a pripájať sa na každý stroj ohľadom aktualizácie systému, TeamCity agentov, Gradle a Maven závislosti pre rýchlejšie zostavenie rôznych projektov.

7.2 Zhodnotenie odbornej praxe

Počas tejto bakalárskej praxe som sa veľa naučil a získal veľa nových vedomostí. Najviac som si prehĺbil znalosti v jazykoch JavaScript a Python, ktorý som využíval takmer počas celej doby praxe. Naučil som sa pracovať v operačnom systéme Linux a používať príkazový riadok ako kľúčový bod k ovládaniu celého operačného systému. Pretože som počas celej doby praxe používal k svojím zdrojovým kódom systém Git, naučil som sa s ním pracovať a používať jeho webové rozhranie a taktiež pracovať s ním pomocou príkazového riadku.

Prax mi dala nové vedomosti, o ktorých som pred absolvovaním nemal žiadnu skúsenosť ani informácie. Zistil som, ako prebieha vývoj rozsiahlých softwarových produktov v praxi a aké nástroje sú k tomu potrebné. Prepojenie TeamCity a virtualizácie mi prinieslo nový pohľad na zostavovanie rôzných produktov z hľadiska efektivity a bezpečnosti.

Literatúra

[1] CHACON, Scott. Pro Git. New York: Distributed to the book trade worldwide by Springer-Verlag, c2009. Expert's voice in software development. ISBN 9781430218340.

[2] Django. The Web framework for perfectionists with deadlines [cit. 09.04.2019]. Dostupné z: https://www.djangoproject.com/

[3] BANKS, Alex a Eve PORCELLO. Learning React: functional web development with React and Redux. Sebastopol, CA: O'Reilly Media, 2017. ISBN 9781491954591.

[4] NICKOLOFF, Jeff. Docker in action. Shelter Island, NY: Manning Publications, Co., [2016]. ISBN 978-1-63343-023-5.

[5] What Is vSphere? - dummies. dummies - Learning Made Easy [cit. 09.04.2019]. Dostupné z: https://www.dummies.com/programming/networking/what-is-vsphere/

[6] JFrog Artifactory | Continuous Delivery Map. Assessment Tools | CA.com [cit. 09.04.2019]. Dostupné z: https://assessment-tools.ca.com/tools/continuous-delivery-tools/en/artifact-repository-management/jfrog-artifactory

[7] Getting Started with TeamCity - Confluence. [cit. 09.04.2019]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Getting+Started+with+TeamCity

[8] What is Gradle? [cit. 10.04.2019]. Dostupné z: `https://docs.gradle.org/current/userguide/what`$_i s_g radle.html 1_g radle_i s_{ag} eneral_p ur$ $//docs.gradle.org/current/userguide/what_i s_g radle.html 1_g radle_i s_{ag} eneral_p urpose$